Yazılım Laboratuvarı Multi Thread Asansör Sistemi

Özge POYRAZ
Kocaeli Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği
180202025@kocaeli.edu.tr

Burak Can TEMİZEL
Kocaeli Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği
180202024@kocaeli.edu.tr

***Özet*—Bu çalışmada çoklu threadler, nesneye yönelik programlama paradigması, arayüz ögeleri ve çeşitli veri yapıları ile bir asansör sistemi gerçekleştirdik. Veri yapıları üzerinde senkron bir şekilde, farklı threadler üzerinden çeşitli işlemler gerçekleştirdik ve dinamik bir arayüz üzerinden elde edilen çıktıları sunduk.**

Anahtar kelimeler—Thread, Multithread Asenkron Programlama, Asansor Sistemi, Thread Safe Veri Yapıları

I. GIRIŞ VE PROBLEM TANIMI

Gerçekleştirdiğimiz proje her ne kadar bir alışveriş merkezi asansör sistemi olarak gözükse de asıl nokta aynı zamanda çalışan farklı threadler üzerinden senkronize bir şekilde sistemle etkileşime girebilmekti. Bu etkileşimler belli kuyruk ve liste veri yapıları üzerinde elemanları ekleme, silme, bölme ve değiştirme gibi threadler arasında olası çakışmaların yaşanabileceği eylemlerdi. Bu işlemleri gerçekleştirirken threadlerin senkronize bir şekilde çalışması kadar veri yapılarının da uygunluğu oldukça önemliydi. Karşılaşılabilecek olası problemlerin çözümü için spesifik yöntemler uygulanması gerekiyordu. Bu sistemi gerçekleştirirken projeyi farklı kısımlara ayırmak faydalı olacaktı. Program mimarisi genel olarak çoklu threadler ve senkronizasyon, asansör sistemi, kullanılan standart ve thread safe veri yapıları ile arayüzden oluşmaktadır. Raporun ilerleyen kısımlarında bu kısımların teknik özelliklerine ayrıntılı bir şekilde tekrar değineceğiz.

İlk etapta farklı amaçlara hizmet eden threadleri oluşturduk. Nesnelerimizi soyutlayacak sınıfları yazdık ve gerekli veri yapılarını bu sınıflar içerisinde tanımladık. Daha sonrasında threadlerin ilgili işlemleri gerçekleştirecek metotlarını yazdık. Threadlere gereken öncelikleri atadık. Veri yapılarını thread safe bir hale getirdik ve synchronized anahtar kelimesiyle aradaki senkronizasyonu sağladık. Daha sonrasında da yine bir thread olan arayüz ile sistemden gerekli çıktıları elde ettik.

II. YÖNTEMLER, YAKLAŞIMLAR VE PROGRAM MIMARISI

Bu kısımda programın farklı özelliklerini oluşturmak için kullandığımız araçlar ve yöntemler üzerinde durularak ayrıntılı olarak bilgi verilecektir. Program mimarisi daha detaylı bir şekilde açıklanacaktır.

A. Çoklu Threadler ve Senkronizasyon

Projede gerçeklenmesi gereken bazı threadler bulunmaktaydı bunlar AvmGiris, AvmCikis, Asansor, Kontrol ve bizim eklediğimiz TestArayuz threadleriydi. Öncelikle her bir thread için Runnable Interface'ini implement ederek gerekli thread objesini oluşturduk. Program çalışmaya başlamadan önce threadler arasında sırasıyla AvmGiris, AvmCikis, Kontrol, Asansor1, Asansor2, Asansor3, Asansor4, Asansor5 ve TestArayuz olacak şekilde bir priority belirledik. Daha sonrasında arayüz üzerindeki başlatma butonu üzerinden threadleri startlayarak override ettiğimiz run fonksiyonlarını giriş noktası olarak çalıştırdık ve her threadimiz çalışmaya başladı. Bu noktada threadlerin sürekli çalışmasını sağlamak için sonsuz bir while döngüsüne aldık ve gerekli zamanlama işlemleri için threadi belli süreleri baz alarak uyku durumuna getirdik. Fakat bu noktada herhangi bir senkronizasyon olmadığı için sistem üzerindeki veri yapılarına müdahalelerde bazı problemler ortaya çıkmaktaydı. Synchronized anahtar kelimesiyle veri yapılarına müdahale edilen yerlerde senkronizasyonu sağladık. Threadler sahip oldukları farklı işlevlerini yerine getirmekteydi. Örneğin AvmGiris sınıfındaki yeniInsanEkle metodu belli bir zaman aralığında çalışarak zemin kuyruğuna yeni insan gruplarını ekliyordu. AvmCikis metodu da benzer bir şekilde sürekli olarak çalışıp katlardanCikisYaptirt metoduyla rastgele katlardan rastgele sayıda insanları çıkmak için gerekli katların kuyruğuna ekliyordu.

Asansör sınıfı daha sonrasında kendine ait başlığında açıklayacağımız birtakım fonksiyonlarla hareketini gerçekleştiriyor ve çeşitli veri yapılarıyla etkileşime giriyordu. Bu noktada birden fazla asansör olduğu için ve aynı veri yapılarına müdahale ettikleri için senkronizasyonun sağlanması şarttı. Kontrol sistemi de proje dokümanında belirtildiği şekilde kuyruklardaki toplam kişi sayısını baz alarak asansörlerin aktiflik, pasiflik durumlarını değiştiriyordu bunu yaparken o da diğer threadler gibi sürekli olarak çalışıyordu. Proje dokümanında verilenlerin haricinde Arayüzümüz de kendi başına bir threaddi ve bizim belirlediğimiz bir zaman aralığında bize programın o anki çıktısını anlık olarak gösteriyordu ve o da ortak verilere eriştiği için senkron fonksiyonlar yardımıyla çalışıyordu.

B. Asansör Sistemi

Her ne kadar proje çok sayıda thread içerse de kuşkusuz senkronizasyonun ve mimarinin en önemli olduğu nokta Asansör sınıfıydı. Çünkü buradan türetilmiş ve thread olarak çalışan her obje sürekli olarak aynı veri yapılarına müdahale ediyordu ve bu da programın çeşitli exceptionlar döndürmesine sebep oluyordu. Bu sınıfta yapılan işlemlerde çeşitli yöntemler izledik. Bunların bazılarına kullanılan veri yapılarında değinilecektir. Exceptionlardan kaçınmak için bazı yerlerde yapılar üzerindeki gezinmemizi direkt olarak döngü işlemleriyle değil iterator nesneleriyle gerçekleştirdik. Bunların haricinde asansör sistemi belli bir hareket algoritmasına sahiptir ve bunu takip ederek sürekli hareketini gerçekleştirirken aynı zamanda katlardaki yolcularla etkileşime girmektedir. Tüm bunların yanında sürekli olarak Kontrol sınıfı üzerinden de müdahaleye uğramaktadır.

Asansor threadimiz temel olarak sürekli bir hedef belirleme, yolcu indirme, yolcu alma metodlarını çalıştırmakta ve belirli olan duraksama zamanını geçirmekteydi. Nasıl çalıştığını anlamak için öncelikle

hareket mekanizmasını incelememiz gereklidir. Sistemimiz bir alışveriş merkezi baz alınarak çalıştığı için biz yönlü asansör sistemini gerçeklemeyi uygun bulduk. Bu sistemde kişiler çıkmak istedikleri katı değil yönü seçerler ve asansör içerisinde gitmek istedikleri katı seçerler. Bizim programımızda kişilerin hedef katları zaten başlangıçta bellidir. Buradaki kritik nokta asansörün yönü yukarı doğru iken alt katta bekleyen birinden ziyade öncelik olarak yönünün olduğu kattakileri baz almasıdır. Bunu göz önüne alarak bir hedef belirleme metodu yazdık.

Hedef belirleme metodumuz ilk adımında asansörün içerisindeki kişilerin gitmek istedikleri katlara bakar ve yönü yukarı iken en küçük katı seçer ve ilgili kata hareketi gerçekleştirir. Hedef dinamik olarak sürekli belirlendiği için ilgili kata geldiğinde bir sonraki hedefe yönelir. Asansör içerisindeki kişiler tamamen indiğinde hedef belirleme fonksiyonu üst katlara bakar ve çıkmak isteyen yolcu var ise üst katlara hareketi gerçekleştirir. Eğer yoksa yönü aşağı doğru değişir ve aşağı harekete başlar. Eğer en üst kata çıkmışsa yine aynı şekilde yönü aşağı doğru değişir ve aşağı harekete başlar. Yön aşağıyken de benzer bir algoritma izlenir fakat proje kapsamında kişiler üst katlardan sadece zemin katlara hareket ettikleri için bu kısım gözlenebilir değildir.

Asansöre yolcu alma fonksiyonumuz, katlardaki kişi sayısını kontrol ederek işlemi gerçekleştirip gerçekleştirmeyeceğinin ön değerlendirmesini yapar, katlardaki çıkış kuyruklarında kişi var ise asansördeki boş yeri değerlendirir ve boş yer olması durumunda boş yer kadar kişiyi asansöre alır. Burada çeşitli kuyruk işlemleri gerçekleştirir ve düğüm olarak kullandığımız Grup sınıfı objeleriyle işlemler yapar.

Asansörden yolcu indirme fonksiyonumuz da benzer şekilde sürekli olarak asansör içerisindeki kişileri gezerek hedef katlarını, şu an bulunulan mevcut kat ile kıyaslar ve eğer hedeflenen katlara geldiler ise onları asansörden indirir.

Asansör sistemimiz yukarıda genel olarak bahsettiğimiz yöntemleri ve mimarileri kullanarak işlemlerini gerçekleştirmektedir.

C. Kullanılan Standart ve Thread Safe Veri Yapıları

Projenin örnek çıktılarına baktığımızda belli thread çalışma zamanlarında sisteme dahil olan insanların tuple benzeri yapılarla gruplar halinde bulunduğunu gördük. Bizden istenen isterin de bu yönde olmasından dolayı kendimiz bir düğüm yapısı oluşturduk. Bu yapı Grup adını taşımakta ve 2 adet değişken barındırmaktaydı. Bunlar kişi sayısı ve hedef kat değişkenleriydi. İşlemlerimizi bu Grup düğümünü kullanan veri yapıları üzerinde gerçekleştirdik. Grup en temel yapımız olacaktı.

Öncelikle standart kuyruklarla yaşadığımız çeşitli thread sorunlarından kaçınmak için thread safe olan tipleri kullanmaya karar verdik. Bu konuda biraz araştırma yaptıktan sonra tüm kuyruk veri yapılarımızı LinkedBlockingQueue veri yapısıyla değiştirdik ve işlemlerimizi bu yapı üzerinden gerçekleştirdik. Her ne kadar katlardaki kuyruklarda çeşitli işlemler gerçekleştirsek de projede bazı noktalarda listeleri de kullanmamız gerekiyordu örneğin asansörün içerisinde ve katların kuyruk dışında kalan kısımlarında. Normal listeler ve array listler ile de çeşitli thread problemleri yaşayabileceğimiz için yine bir thread safe veri yapısı olan CopyOnWriteArrayList'I kullanmaya karar verdik ve projemize uyguladık. Farklı thread safe yapıları görebilmek adına katlardaki kuyruklarda bu yapıyı kullanırken asansör içerisinde Collections sınıfında bulunan synchronizedList yapısını kullandık

D. Arayüz

Sistem içerisindeki threadler sürekli işlemler gerçekleşiyor ve ilgili yapılar sürekli olarak değişikliğe uğruyordu. Bu yapıların anlık olarak görüntülenmesi için bir kullanıcı arayüzü oluşturduk. Arayüzü netbeans'in form editörü aracılığıyla oluşturduk. Karanlık tema için DarkLaf Look and Feel'ini projemize ekledik. Arayüz şablonu oluştuktan sonra artık anlık olarak değerleri alıp çıktı olarak renderlamamız gerekiyordu. Bu noktada arayüzümüzü de bir threade çevirdik. Belli zaman aralıklarında tüm veri yapılarını okuyarak ilgili değerlerini alıp komponentler üzerinde görüntülüyordu ve bunun için bir takım yardımcı fonksiyonlar kullanıyordu.

Yardımcı fonksiyonlar genel olarak iteratorler aracılığıyla veri yapılarını okuyarak onları bir string objesine dönüştürüyor ve daha sonrasında bu string objesini döndürüyordu. Arayuz de ilgili kısımlarda bunları görüntüleyerek kullanıcıya çıktıyı sunuyordu.

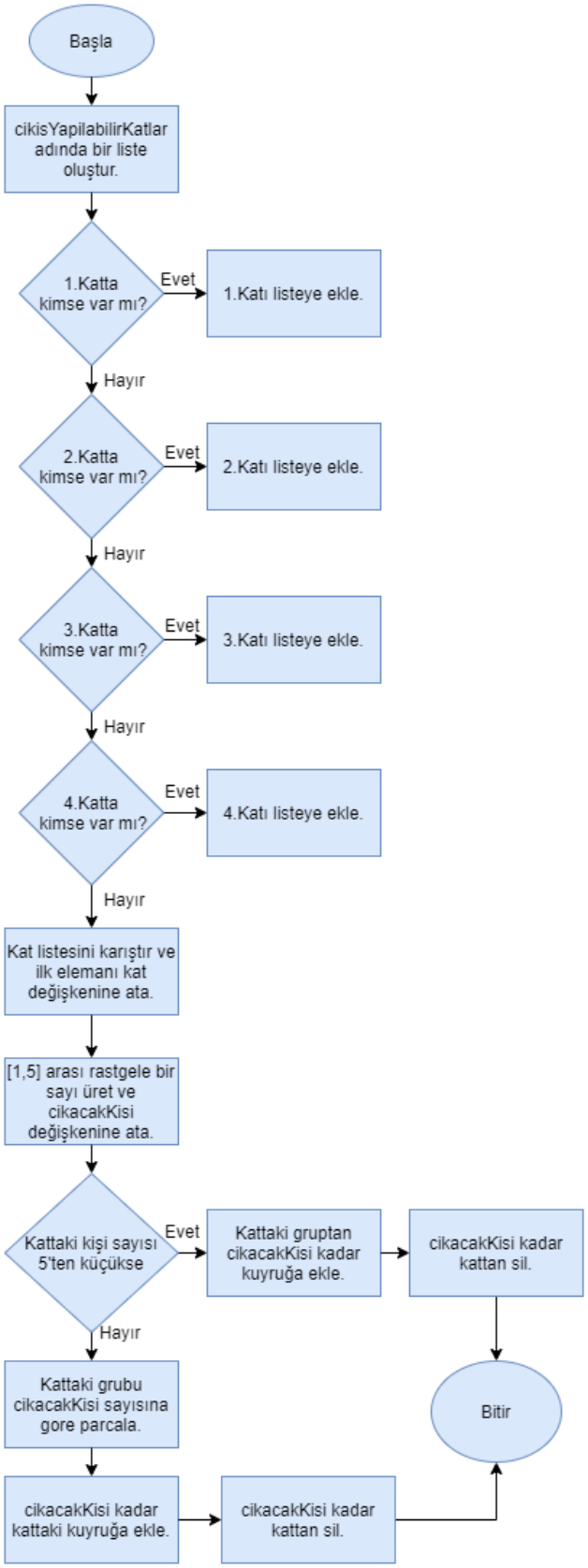
III. KAZANIMLAR

Projeyi gerçekleştirirken bir çok kazanım elde ettik. Diğer gerçekleştirdiğimiz projelerimizde de çeşitli amaçlarla thread kullanmıştık fakat genelde kullandığımız bu threadler farklı görevleri gerçekleştirmek üzerine çalışıyordu. Bu sefer ise ortak işler gerçekleştiren threadlerle karşılaştık ve bu da beraberinde bir çok problem ortaya çıkardı. Bunları çözerken threadlerin genel işleyişi, senkronizasyon ve bunu sağlayan anahtar kelime, blok gibi konularda araştırmalar yaparak çeşitli bilgiler edindik. Ayrıca çeşitli thread safe veri yapılarını ve multithread çalışırken karşılaşılan çeşitli sorunların önüne nasıl geçtiklerini öğrenerek deneyimledik.

IV. KARŞILAŞTIĞIMIZ SORUNLAR, ÇÖZÜMLER VE EKSIKLIKLER

Threadler arasındaki senkronizasyonu tam sağlamadan önce direkt olarak proje iskeletini kurmaya çalışmamız ve çıktıyı görüntüleme isteğimiz ile beraber çeşitli thread exceptionlarıyla karşılaştık. Bunların hepsini yukarıda bahsettiğimiz çeşitli yöntemler ile gidersekte bu bize bir takım zaman kaybı yaşattı. Veri yapıları içerisinde döngüler ile gezerken threadlerin sürekli dinamik veri yapılarının boyutlarıyla oynaması, yapıyı gezinen bir diğer thread de problemler çıkarmaktaydı. Burda ilk etapta problem döngüyü tersten dönerek, işlem bittiğinde kırmak gibi çeşitli şekillerde gidermeye çalıştık. Problemi çözmüş olsakta doğru bir yöntem olmadığını düşünerek bu konuda uygun yöntemin veri yapılarını iterator nesneleriyle dönmek olduğunu öğrendik. İyi bir çözümle kalıcı olarak bu problemi de gidermiş olduk. Proje içerisinde karşılaştığımız ve bir eksiklik olmasa da kafa karışıklığı yaratan kontrol threadinin asansörleri, aktif pasif hale getirirken örneğin kuyruk sayısı üst limiti geçtiğinde , sürekli çalıştığı için bir asansörü açtığında tekrar limitin üstünde mi diye bir değerlendirme zamanına ihtiyaç duymasıydı isterlerde net olarak belirtilmediği için burada kendimiz bir çözüm bulmak zorunda kaldık.

V. AKIŞ DIYAGRAMI VE YALANCI KOD



AvmCikis Threadinin Genel Çalışma Algoritmasının Akış Şeması

```
senkronize fonksiyon
asansoreYolcuAl(ThreadSafeKuyrukVeriYapısı kuyruk,
Asansor asansor):

    Eğer kuyruk boyutu 0'a eşit veya küçük ise:
        fonksiyonu sonlandır ve boş değer döndür

        Döngü(asansordeki kişi sayısı < asansorün kapasitesi
        olduğu sürece):

            Eğer kuyruk boyutu 0'a eşit veya küçük ise:
                fonksiyonu sonlandır ve boş değer döndür

            Eğer kuyruktaki kişi sayısı 0'dan büyük ise:
                kuyruk başı = kuyruk listesinin başındaki grubu
                dondur

                bos yer = asansor kapasitesi - asansordeki mevcut
                kişi sayısı

                Eğer bos yer, kuyruk başındaki kişi sayısından
                büyük eşitse:
                    asansorun içindekilerine kuyruk başını ekle
                    kuyruk başını, kuyruktan çıkart

                    asansordeki mevcut kişi sayısına, kuyruk
                    başındaki kişi sayısını ekle

                Değilse:
                    yeni dugum = bosYer kadar, kuyruk başıyla aynı
                    hedefte yeni bir düğüm oluştur

                    asansorun içindekilere yeni dugum ekle

                    asansorun mevcut kişi sayısından yeni dugumun
                    kişi sayısını çıkart

                    kuyruğun başındaki kişi sayısından boş yer
                    kadar çıkart
```

Yukarıdaki yalancı kod parçası Asansor threadinin asansoreYolcuAl adlı metoduna aittir.

VI. GELIŞTIRME ORTAMI VE KULLANILAN DIL

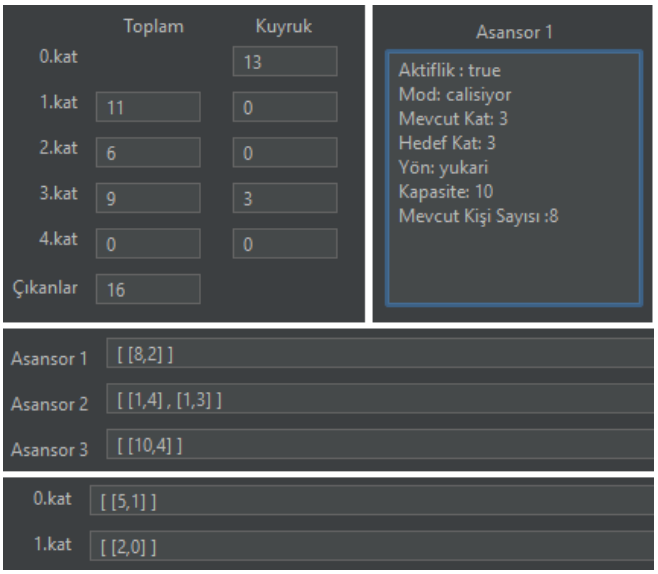
Projeyi Java programlama dilinde Windows işletim sistemi üzerinde gerçekleştirirken, geliştirme ortamı olarak 1.8 JDK konfigürasyonlu Netbeans ve Eclipse idelerini kullandık. Proje Java Maven projesi olup ilgili bağımlılıkları pom.xml dosyasında girilidir.

VII. PROGRAMIN GENEL YAPISI VE TASARIMI

Program çıktıların kullanıcıya sunulduğu tek bir form içeriyor. Sürekli çalışan arayüz anlık olarak elde ettiği verileri son kullanıcıya sunuyor.

.

Programın Genel Tasarımı



Çeşitli Bilgilendirme Panelleri

VIII. Yapılan Araştırmalar

Projeyi gerçekleştirmeden önce birtakım araştırmalar yaptık. Bu kısımda yaptığımız araştırmalardan bahsedeceğiz.

İlk araştırmalarımız threadlerin genel yapısı ve mantığı hakkındaydı(1). Her ne kadar daha önce kullansak da temel noktalarda bilmediğimiz şeyleri öğrendik ve bildiklerimizi detaylandırdık. Farklı tanımlamaları ve kullanım yöntemlerini öğrendik.

Daha sonrasında projeyi gerçeklerken yaşadığımız bazı sorunlar sebebiyle daha fazla araştırma yapmaya karar verdik. Yaptığımız araştırmalardan bahsetmemiz gerekir ise, kuyruk yapılarımız için thread safe bir veri yapısı seçmemiz gerekiyordu bu yüzden LinkedBlockingQueue veri yapısını araştırdık(2)(3). Aynı şekilde array listlerimiz içinde bir yapı bulmamız gerekiyordu burda da CopyOnWriteArrayList(4) ve SynchronizationArrayList(5) yapılarını araştırdık.

Threadler için priority kavramını araştırdık(6). Projenin bazı noktalarında örneğin AvmGiriş gibi önceliği belli olan threadler için faydalı olacağını düşündük.

Proje içerisinde bazı veri yapılarını gezinirken, boyut değişiminden kaynaklı exception sorunları yaşadığımızdan bahsetmiştik. Bunu engellemeye çalışırken iterator sınıfını ve fonksiyonlarını(7) araştırmak zorunda kaldık.

Sistemde tüm mimarimizi kurduktan sonra fonksiyonel hale getirirken methodları senkron hale getirmemiz gerekiyordu. Bu noktada synchronized anahtar kelimesini(8) araştırdık. Methodlar üzerinde kullanımını ve blok halinde kullanımını inceledik.

Tüm bu araştırmalarımızla birlikte kazanımlar bölümünde de bahsettiğimiz tüm kazanımlara sahip olmuş olduk.

IX. Deneysel sonuçlar

Proje boyunca sistemi kurarken, farklı deneysel sonuçlar elde ettik. Öncelikli olarak fark ettiğimiz bir şey, multithread olarak çalışan bir programın içerisinde hata ayıklama işlemi yapmak oldukça zor bir hal alıyordu. Farklı threadlerin farklı zamanlarda çalışması hatanın tespitini oldukça güç bir hale getiriyordu. Bu noktada hata yakalamak için çeşitli yöntemler denememiz ve uygulamamız gerekti. Kimi zaman test amaçlı threadleri durdurmak, uyutmak gibi işlemler ile hataları yakalamaya çalıştık. Try-catch bloklarını daha aktif kullanarak karşılaşmasak bile gelecekte ayıklaması zor olacağı için tüm istisnaları yakalamaya çalıştık.

X. Sonuç

Bu projeyi gerçekleştirerek senkronize bir şekilde çalışan multi thread bir asansör sistemi oluşturmayı başardık. Çeşitli thread işlemleri, asansör algoritması ve veri yapılarını kullandık, arayüz işlemlerini ve isterleri gerçekleştirdik.

XI. Kaynaklar

(1) https://www.tutorialspoint.com/java/java_multithreading.html
(Erişim Tarihi: 1.12.2020)

(2) https://www.baeldung.com/java-queue-linkedblocking-concurrentlinked
(Erişim Tarihi: 3.12.2020)

(3) https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/LinkedBlockingQueue.html
(Erişim Tarihi: 3.12.2020)

(4) https://www.geeksforgeeks.org/copyonwritearraylist-in-java/
(Erişim Tarihi: 3.12.2020)

(5) https://www.geeksforgeeks.org/synchronization-arraylist-java/
(Erişim Tarihi: 3.12.2020)

(6) https://www.studytonight.com/java/thread-priorities-in-java.php
(Erişim Tarihi: 3.12.2020)

(7) https://www.w3schools.com/java/java_iterator.asp
(Erişim Tarihi: 3.12.2020)

(8) https://ufukuzun.wordpress.com/2015/02/26/javada-multithreading-bolum-3-synchronized-anahtar-kelimesi/
(Erişim Tarihi: 3.12.2020)